3134 BEST AVAILABLE COP

XP-000801610

Generalized Inverse Multiplexing of Switched ATM Connections

Fabio M. Chiussi Denis A. Khotimsky Santosh Krishnan

Bell Laboratories, Lucent Technologies
101 Crawfords Corner Road, Holmdel, NJ 07733, U.S.A.
Email: {fabio, dkhotimsky, sk}@lucent.com

Abstract — *Inverse Multiplexing for ATM (IMA) has been standardized by the ATM Forum to provide a high-capacity logical link by grouping several lower-capacity physical links. IMA is intrinsically point-to-point, assumes congestion-free links, and only tolerates relatively constant differential delays between the links. In this paper, we generalize the notion of inverse multiplexing by introducing switching and buffering within the inverse-multiplexed segment. The new scheme, which we call* Switched Connection Inverse Multiplexing for ATM *(SCIMA) allows to implement an $N \times N$ switch with ports operating at rate kR by using a $kN \times kN$ core ATM switch with ports of rate R.*

SCIMA executes demultiplexing on a selected set of virtual connections, dynamically distributes their traffic onto a group of switching paths, and performs egress re-assembly using an asynchronous method for differential delay compensation. The scheme is simple to implement, has low overhead, is robust in presence of cell loss within the switch, handles congestion, and tolerates wide variations in differential delays.

1 Introduction

Inverse multiplexing has been known for quite some time [1] as a technique to implement a single high-capacity logical channel by using several lower-capacity channels, where the distribution and aggregation of the traffic flow to and from the individual channels is transparent to the higher layers [2, 3]. More recently, inverse multiplexing has been applied within the context of Asynchronous Transfer Mode (ATM) networks, and standardized by the ATM Forum in the *Inverse Multiplexing for ATM* (IMA) specification in July 1997 [4]. This specification defines a new protocol-stack entity, called the *IMA sublayer*, which is located between the Physical Layer Transmission Convergence sublayer and the ATM layer, and performs "inverse multiplexing and demultiplexing of ATM cells in a cyclic fashion among links grouped to form a higher bandwidth logical link whose rate is approximately the sum of the link rates."

The IMA protocol is inherently designed to provide a *point-to-point* logical link. As shown in Figure 1(a), the protocol treats the entire cell stream over the logical link as a *single* entity. The protocol segments the cell stream into frames of a specified duration, and sends the frames across a group of N independent links by dispatching the cells to the individual links in a synchronous cyclic order, using special Filler cells to maintain a continuous cell stream at the Physical layer in the absence of ATM layer traffic. The frames are reconstructed at the far end of the IMA logical link using alignment information contained in overhead *IMA Control Protocol* (ICP) cells, which are inserted once per frame on each link. Using the ICP cells, the IMA protocol can compensate for different end-to-end delays on each link, provided that such differential delays stay relatively constant over time; in this way, it is possible for the individual links to take different paths in the network, and even pass through different switches, as long as the end-to-end delay variation is controlled.

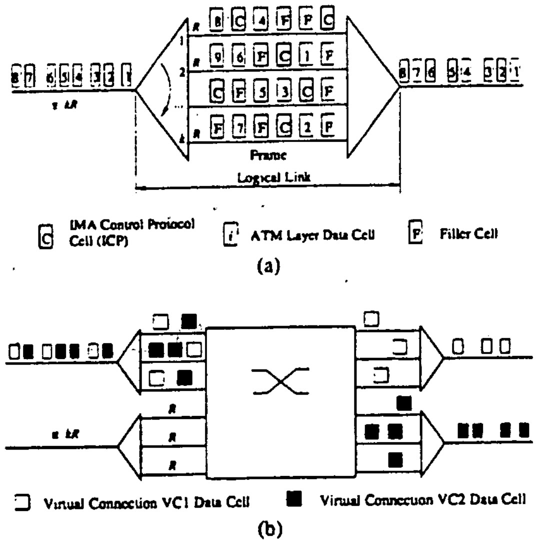
Frame
Logical Link
IMA Control Protocol Cell (ICP)
ATM Layer Data Cell
Filler Cell
(a)
R
Virtual Connection VC1 Data Cell
Virtual Connection VC2 Data Cell
(b)

Figure 1: Inverse Multiplexing: (a) Conventional point-to-point IMA; (b) Proposed Switched Connection Inverse Multiplexing for ATM (SCIMA).

In this paper, we generalize the notion of inverse multiplexing to the context of scaling the port capacity of an ATM switch by grouping a number of ports of a certain available capacity into a "logical" port of capacity approximately equal to the sum of the port capacities. As the market demands switches capable of terminating trunks of higher and higher bandwidth, the issue of scaling the port capacity becomes crucial. Inverse multiplexing can potentially provide an economical way to interface the new higher-rate trunks with existing switches having ports operating at lower rate. It can also become a flexible option in architecting new switches to push the port rate beyond what is possible or cost effective in terms of port processing and port rates in the switching fabric. However, due to its intrinsic point-to-point nature and rather rigid handling of differential delays, conventional IMA is not suitable for this purpose.

We introduce a more general inverse multiplexing scheme, called the *Switched Connection Inverse Multiplexing for ATM* (SCIMA), which removes the limitations of IMA. The context where SCIMA operates is illustrated in Figure 1(b). A core switch with input and output ports of a given capacity R terminates high-capacity transmission trunks. This is accomplished by grouping the input and output ports of the core switch, and associating each group with a controller connected to a high-capacity trunk. The SCIMA scheme operates in the input controllers to distribute the traffic arriving on the incoming trunk among the switching paths originating at that logical port, and in the output controllers to perform aggregation of the traffic onto the outgoing trunk. For example, using SCIMA, currently available OC-12 switch ports [5] can be grouped together to interface to emerging OC-48 and OC-192 trunks.

SCIMA is a significant generalization of conventional IMA in three main aspects. First, SCIMA demultiplexes and aggregates the traffic *separately* for each Virtual Connection (VC) that constitutes the high-rate trunk traffic flow, rather than the entire high-rate stream as a whole; indeed, since SCIMA operates at the edges of a core switch, each VC has to be switched individually, based on its destination, allocated rate, traffic class, and other Quality of Service (QoS) requirements, as opposed to the point-to-point environment where IMA operates. Second, SCIMA handles congestion and the possibility of cell loss; in fact, the switching paths within the switch contain buffers, shared by independent bursty traffic flows, and may experience at times heavy congestion, contrary to the effectively bufferless and congestion-free links in IMA. Finally, SCIMA deals with large, unpredictable differential delays as well as significant delay variations, caused by the rapidly varying cross-traffic and congestion conditions existing along the different switching paths; this requires an asynchronous scheme for cell re-ordering and differential delay compensation, as opposed to the synchronous method used in IMA.

Our SCIMA scheme assumes a very general architecture for the core switch, and is therefore widely applicable. The scheme is simple to implement and requires very low overhead. To keep complexity low, SCIMA performs traffic demultiplexing (which we refer to as *splitting*) of only a selected set of VC's. The cell ordering information consists of a small number of bits, and is transmitted in the local switching header of each cell, in order to support the asynchronous approach of the novel *Cell Ordering and Re-Assembly Protocol* that we use in SCIMA. The per-VC traffic splitting allows to transparently pass the QoS guarantees provisioned by the core switch onto the high-rate trunk. SCIMA relies on the ability of the switch to execute a connection admission policy which, along with accepting a VC, makes an assignment of a split weight vector which specifies how the traffic for that VC should be split on the available switching paths.

We provide a number of mechanisms to make SCIMA robust against cell losses in the core switch. First, we use backpressure to allow persistent cell losses (such as those due to buffer overflow) only at the input controllers and at the output ports of the core switch, where it is easy to guarantee that they do not affect the re-assembly process. Second, we devise a cell re-ordering scheme that is robust to isolated losses (such as those caused by electrical glitches and sporadic errors) occurring at any point in the switch. Third, we provide an *Advanced Frame Recovery Mode* to promptly re-establish correct cell reordering in those situations where, despite the two previous mechanisms, a loss pattern that cannot be handled by the re-assembly process has occurred. Finally, SCIMA copes with congestion in the different paths by using an *Adaptive Cell Dispatching Algorithm* that

re-order →

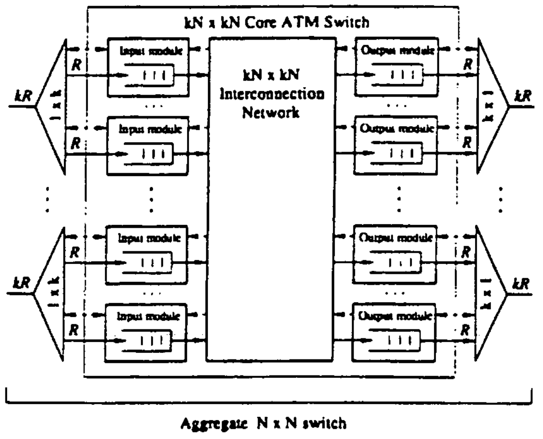

Figure 2: SCIMA Architecture

performs dynamic load balancing and avoids congested paths, as made possible by the asynchronous nature of the scheme.

The issue of cell re-assembly at the output of the switch that arises with inverse multiplexing in an ATM switch has been addressed in the context of multi-path switch architectures. However, the existing solutions are based either on sending sequence numbers in the cell headers, which is costly in terms of overhead [6, 7], or on an enforced delay equalization for specific input-output port pairs [8, 9] which effectively precludes QoS provisioning to the individual VC's in the inverse-multiplexed flow.

The rest of this paper is organized as follows. In Section 2, we define the underlying switch architecture assumed by SCIMA. In Section 3, we motivate the per-VC traffic splitting approach, provide an overview of the SCIMA scheme, and discuss our strategy to handle cell loss. The individual functional components of SCIMA are discussed in Sections 4 through 6. Finally, Section 7 provides concluding remarks.

2 Model

We begin with the description of the underlying switch architecture. As shown in Fig. 2, the $kN \times kN$ core ATM switch has a very general mutistage architecture, which consists of kN *input modules* and kN *output modules* communicating with each other by means of an *interconnection network*. Buffers may be provided in both input and output modules as well as in the center-stage interconnection network.

The ports of the core switch, each of capacity R bits/sec, are partitioned into groups of size k. Each group is connected to a single inverse multiplexing *controller* handling a high-rate trunk of capacity kR. The core switch together with the input and output controllers form therefore an $N \times N$ *aggregate switch* with ports of capacity kR bits/sec. Each input controller acts as a $1 \times k$ demultiplexor, and distributes the traffic from the high-rate input trunk to the ports in its corresponding group. The demultiplexed traffic flows in each input port are treated independently by the core switch and follow separate switching paths within the switch to the destination egress ports. The output controller acts as a $k \times 1$ multiplexor, and consolidates the traffic arriving from different egress ports in the same group, while maintaining cell order in the aggregated flow.

Control over buffering and cell loss within the aggregate switch is attained through the use of backpressure, which may be applied at five different locations, as discussed in the next section: from the output controllers to the core switch output modules, from the output modules to the output of the interconnection network, from the outputs to the inputs of the interconnection network, from the interconnection network to the input modules; and from the input modules to the input controllers.

3 SCIMA Overview

3.1 Splitting of Virtual Connections

The obvious primary objective of SCIMA is to deliver the traffic of each VC arriving at the inputs of the aggregate switch to its destination, while maintaing the cell order for that VC. When the required bandwidth of the VC is higher than the capacity of the ports in the core switch, inverse multiplexing of the traffic of that VC across the core switch must be performed. We refer to this operation as *splitting* the VC, meaning that the traffic flow for that VC is split at the input controllers into independent subflows (which we call *subconnections*), each sent to a different input port of the core switch and treated independently by the core switch; the subconnections are reassembled at the output controllers into an aggregate flow that maintains the cell order of the original flow.

The alternative approach to splitting at the VC level would be splitting the *entire* flow associated with each pair of input and output ports in the aggregate switch among the available switching paths. Although this approach can achieve almost ideal pooling of the switching capacity and load balancing over the switching paths, it cannot provide differentiated Quality of Service to traffic classes and individual VC's within a single flow. Splitting at the VC level makes it possible to transfer transparently the QoS properties of the core switch to the aggregate switch.

While it is clear that VC's of bandwidth higher than the capacity R of the ports in the core switch *have* to be split, it may be desirable to also split VC's of bandwidth lower than R in order to ensure load distribution and balancing within the set of the switching paths which exist between each input and output pair in the aggregate switch. The trade-off here is between increased utilization and reduced call blocking probability, on one hand, and complexity and overhead to re-assemble the split VC's, on the other. We have observed that a relatively small number of split VC's is sufficient to guarantee good utilization and blocking probability in most circumstances (a complete performance characterization of the aggregate switch as a function of the number of split VC's is the topic of a forthcoming paper). In order to keep implementation complexity and overhead low, SCIMA uses this observation and restricts splitting to a selected set of VC's, while routing the others through individual switching paths. In other words, the number of admitted split VC's is treated as a special kind of manageable resource (similar to line bandwidth and buffer space), which has to be taken into consideration by the local Connection Admission Control (CAC) function.

In summary, SCIMA provides the capability of splitting a (relatively small) number S of VC's; splitting of a VC that is admitted by CAC is performed in the following circumstances:

- If the bandwidth B requested by the VC exceeds the port rate R of the core switch ($R < B \leq kR$), then the VC is split. There may be at most $k - 1$ such VC's that can be admitted (again, this is the minimum number of VC's that SCIMA must be able to split; $k - 1$ of the S VC's that SCIMA can split are always reserved for this purpose).
- If the bandwidth B requested by the VC is lower than R, but the residual switching capacity of any available switching path is not sufficient to accommodate the VC, while the request can be granted if the residual capacities of several paths are pooled, the CAC may decide to accept the VC by splitting it.
- If the bandwidth B requested by the VC is lower than R and can be accommodated by a single switching path, but admission without splitting is likely to create a highly uneven load distribution among the different switching paths, thus increasing the differential delay for the existing VC's, the CAC may decide to split that VC.

3.2 Load Distribution

The distribution of load within the aggregate switch is performed both statically and dynamically by SCIMA. Static load distribution occurs at the connection admission stage.

Let C_r be a connection request arriving at the high-rate input trunk s which has to be routed to the high-rate output trunk d. The CAC strategy (the details of which are outside the scope of this paper) is based on the currently allocated resources and the traffic descriptor of C_r. If the admission decision is positive, a *split weight vector*

$$W^{(r)} = \left(w_1^{(r)}, w_2^{(r)}, \ldots, w_n^{(r)}\right),$$

such that $\sum_{i=1}^{n} w_i^{(r)} = 1$, is assigned to C_r; n is the number of available switching paths between the s-th ingress port group and the d-th egress port group. (If C_r is admitted without splitting and routed through a single switching path i, then $w_i^{(r)} = 1$.) In order to achieve a strictly-non-blocking aggregate switch, n should be equal to k^2; however, in practice, we have observed that imposing the restriction of a single switching path per port, when used with proper CAC, does not severely compromise the blocking properties. Therefore, it is often sufficient to assume $n = k$.

SCIMA distributes the traffic of each split VC to the switching paths in proportion to the specified split weights. In addition, to handle congestion, SCIMA also performs dynamic load re-distribution in the course of regular switch operation, using backpressure information from the input modules in the core switch, as discussed in Section 6. Due to the asynchronous approach in the cell re-assembly protocol, SCIMA can avoid to send traffic to a path that is experiencing congestion, and redirect it to another path.

3.3 Confronting Cell Loss

In this subsection, we outline the strategy that we use in SCIMA to handle cell loss in the switch.

For our purpose, we categorize cell loss into two classes: preventable and inevitable. The *preventable* losses are associated with a specific switch functionality and can, at least in principle, be avoided by proper traffic engineering, control policy, or switch architecture. This class is represented, for example, by the cell loss connected to buffer overflow or usage parameter control (policing). These losses may occur in isolation or in bursts affecting many consecutive cells. The *inevitable* losses are caused by uncontrollable stochastic external factors. These are exemplified by random electrical glitches leading to cell delineation errors, header corruption, mis-routing, etc. Typically, most of the inevitable losses are *isolated*, i.e., only a single cell is affected within a relatively large time interval. Occasionally, such a type of loss may be *persistent* and affect more than one cell in a row.

The strategy that we use in SCIMA aims at making the scheme robust against all types of cell loss while keeping the overhead low. We achieve this objective by restricting where the preventable losses occur for the split connections, making the scheme able to handle single losses at all points in the switch, and providing a way to recover from the occasional inevitable losses that are persistent.

As far as preventable losses are concerned, we can control where they occur through the use of backpressure. Specifically, we restrict the preventable losses to occur only where it is simple for SCIMA to detect and recover from them, namely, in the input and output controllers and in the output ports of the core switch. We guarantee that no cell is lost in the center-stage interconnection network by applying backpressure to the input modules of the core switch. Similarly, we guarantee that no cell for the split connections is lost in the input modules by applying backpressure (selectively for the split connections) to the input controllers. In addition, again only for the split connections, such functions as policing and Early/Partial Packet Discard are disabled or transferred to the input controllers. We allow losses for the split connections at the output ports of the core switch. In this case, it is relatively simple to detect a loss and report it to the output controller. The loss report contains the SCIMA protocol portion of the discarded cell's local header (described in Section 4 below), so that the egress re-assembly can proceed uninterrupted even when the cell itself has been dropped.

Contrary to preventable losses, the locations in the switch where inevitable losses may occur cannot be controlled. However, as mentioned above, such inevitable losses occur with high probability in isolation. For this reason, we have made SCIMA robust to isolated losses anywhere in the switch.

Obviously, persistent inevitable losses, although not very likely, cannot be excluded. Consequently, we have made SCIMA capable of *detecting* a persistent loss, up to a certain maximum duration (the details are discussed in Section 4 below). When the scheme detects a persistent loss, cell re-ordering is momentarily suspended, and an Advanced Frame Recovery Mode is invoked to resynchronize the re-assembly process. If the duration of the persistent loss is larger than the maximum duration recognized by SCIMA, it is possible that the scheme cannot detect the loss or that it cannot resynchronize. However, in a practical switch, other means are typically provided to detect very long persistent inevitable losses, such as those due to permanent faults, and properly communicate the event to the various hardware components in the switch.

3.4 Functional Components of SCIMA

Once a split VC is admitted, the *Cell Ordering and Re-assembly Protocol* is instantiated for that VC om the corresponding pair of input and output controllers. This protocol, described in Section 4 below, is responsible for the re-assembly of the VC flow by exchanging cell ordering information, which is passed transparently from the ingress to the egress of the switch in the cell's local routing header. The protocol is consistent with the cell loss handling strategy described above, and is therefore robust against isolated losses, as well as able to detect persistent losses. When a persistent loss is detected, the *Advanced Frame Recovery Mode* described in Section 5 below is invoked.

Finally, each input SCIMA controller performs the slot-by-slot cell dispatching of the admitted ATM VC's, both split and unsplit, to the switching paths of the associated core switch port group. The execution of this task is ensured by the *Adaptive Cell Dispatch Algorithm* described in Section 6, which is able to dynamically redirect a split flow destined to a congested path to a different path.

4 Cell Ordering and Re-assembly Protocol

4.1 Problems with Re-Ordering

Before describing the protocol itself, we identify some difficulties involved in designing a suitable split-connection cell re-ordering scheme. Here, we are interested in devising a scheme that requires low overhead and is robust to isolated losses. In addition, the scheme must work properly regardless of how the cells of a split VC are actually distributed among the possible paths, and not rely on all the paths to be used at a given time. This is necessary for our dispatching algorithm to be able to redirect flows around paths that are experiencing congestion.

A solution that has been proposed in the past for cell re-assembly [6, 7] is to transmit a sequence number in the cells' local header. The output controller then buffers the cells of each subconnection in a First-In-First-Out (FIFO) queue, and reconstructs the flow by serving the cells out of the subconnection FIFO queues by increasing order of their sequence number.

The problem with this solution is that, in order for it to work properly, the period with which the sequence number repeats (the sequence number consists of a finite number of bits in the local header of the cells) must be longer than the largest number of cells that can simultaneously be transiting the switch on different subconnections. This implies that the differential delays between subconnections be bounded and, for the overhead to be low, that such a bound be relatively small. Unfortunately, this is in general not the case in current switches, where the buffer capacities from an input to an output may be of the order of several million cells, and where cells are not necessarily served in a FIFO order but according to more complex QoS scheduling. The problem caused by a period of the sequence number that has insufficient length is that more than a single cell with the same sequence number can simultaneously be in transmission, leading to the possible scenario illustrated in Fig. 3: when, due to congestion, the cell dispatch procedure temporarily skips some of the switching paths (path $P0$ in the example), the cells at the head of two or more subconnections FIFO queues in the re-assembly buffers may have conflicting sequence numbers.

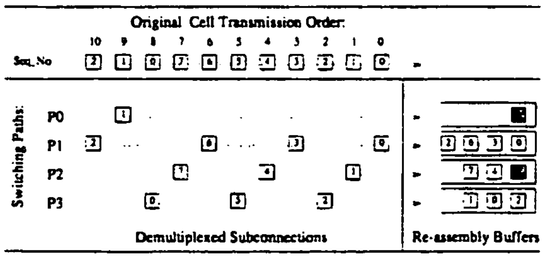

Figure 3: Sequence numbers causing re-assembly failure.

Complementing the total order sequence number with a PREV_PATH pointer which identifies the switching path to which the *preceding* cell of the same split VC was dispatched does not present an improvement. For example, in the scenario of Fig. 4, the same two cells which caused confusion in the previous example would still remain indistinguishable.

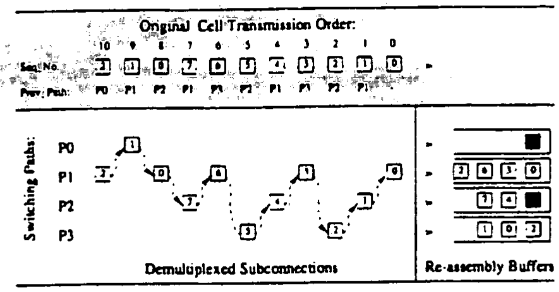
Original Cell Transmission Order:
Seq_No
Prev_Path:
Switching Paths:
P0
P1
P2
P3
Demultiplexed Subconnections
Re-assembly Buffers

Figure 4: PREV_PATH pointer in combination with sequence numbers causes re-assembly failure.

Transmission of a per-VC sequence number along with a NEXT_PATH pointer, which identifies the *subsequent* switching path, allows to remove ambiguity in the re-assembly process. However, when even a single cell is lost in transmission, this approach may fail to detect it. Such a scenario, which involves two cells with identical sequence numbers being transmitted over the same switching path, is shown in Fig. 5.

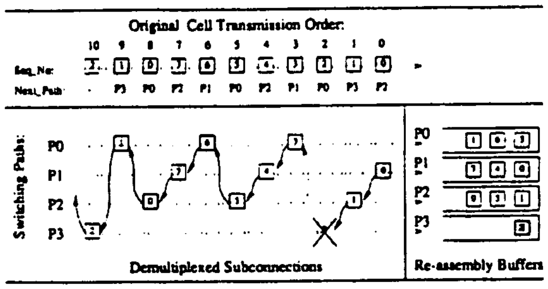
Original Cell Transmission Order:
Seq_No
Next_Path
Switching Paths:
P0
P1
P2
P3
Demultiplexed Subconnections
Re-assembly Buffers

Figure 5: NEXT_PATH pointer in combination with sequence numbers let a cell loss go undetected.

4.2 Description of the Protocol

Our Cell Ordering and Re-assembly Protocol takes advantage of the NEXT_PATH pointers combined with the use of *per-subconnection* (rather than per-VC) sequence numbers. In addition, to achieve robustness in the event of a single cell loss, each NEXT_PATH value is transmitted twice: with its proper cell and in the header of the subsequent cell of the same subconnection. Figure 6 shows the format of the SCIMA portion of local routing header. K is the number of switching paths in a group, S the number of simultaneously supported split VC's, and M the sequence number modulus.

logS	...	2	1	logM	...	2	1	logK	...	2	1	logK	...	2	1
Split_ID				Seq_No				Next_Path				Prev_Next_Path			

Figure 6: SCIMA protocol fields of the cell's local header.

The $Split\_ID$ field is actually not part of the SCIMA overhead, since it is accommodated in the connection identifier field that is anyway present in the local switching header. The sequence number modulus M depends only on the duration of the persistent inevitable losses that SCIMA needs to detect. As mentioned above, since in a practical switch other means are provided to detect very long persistent inevitable losses, a small value of M is typically sufficient.

The input SCIMA controller maintains the following data for each split VC:

- a pointer P_{next} to the switching path scheduled for the next cell transmission;
- a vector $N_{send}[1:K]$ indicating the sequence number of the next cell for each of the K subconnections;
- a vector $P_{prev}[1:K]$ of the P_{next} pointers that were transmitted with the most recent cell of each of the K subconnections;

Two registers $CRNT$ and $PRED$ are used for the temporal storage of the current switching path and the predicted switching path, respectively.

On admission of a split connection, P_{next} is initialized to a pre-negotiated value, $N_{send}[1:K]$ are all zeros, and $P_{prev}[1:K]$ are irrelevant. Subsequently, when a cell belonging to the given split VC is being transmitted, the input controller takes the following steps:

(I1) copies the current value of P_{next} to register $CRNT$ and uses the cell dispatch algorithm to identify the switching path to be used for the transmission of the next cell of the same VC, storing this value in $PRED$;

(I2) modifies the information fields of the cell's local routing header (refer to Fig. 6) as follows:
$Seq\_No \leftarrow N_{send}[CRNT]$;
$Next\_Path \leftarrow PRED$;
$Prev\_Next\_Path \leftarrow P_{prev}[CRNT]$;

(I3) updates the stored P_{prev} vector:
$P_{prev}[CRNT] \leftarrow PRED$;

(I4) increments the sequence number of the switching path in use:
$N_{send}[CRNT] \leftarrow (N_{send}[CRNT]+1) \bmod M$;

(I5) stores the identity of the next path to be used for the given split VC:
$P_{next} \leftarrow PRED$;

(I6) dispatches the cell to the switching path identified by the register $CRNT$.

On the receiving side, the output SCIMA controller maintains, also on a per split connection basis, the following data structures:

- a set $Q_{FIFO}[1:K]$ of re-assembly buffers with FIFO service discipline;
- a pointer P_{exp} to the switching path on which the next cell of the given split VC is expected;
- a vector $N_{recd}[1:K]$ indicating the expected sequence numbers of the cells from the respective subconnections.

On connection admission, P_{exp} is initialized to the same pre-negotiated value as P_{next} at the ingress. The sequence numbers $N_{recd}[1:K]$ are all set to zero. When a cell arrives on a switching path p, it is enqueued in $Q_{FIFO}[p]$.

The output SCIMA controller performs the following steps:

(O1) periodically checks the re-assembly FIFO indexed by P_{exp}, and, when it becomes occupied, inspects the sequence number Seq_No of its first cell;

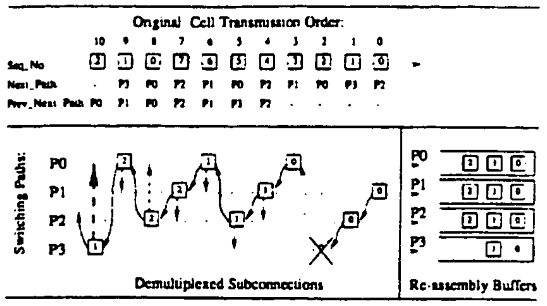

Figure 7: Per-subconnection sequence numbers in combination with NEXT_PATH pointer of the current and the previous cells of the same subconnection.

(O2) if Seq_No matches $N_{rcvd}[P_{exp}]$, then the expected sequence number and switching path index are updated as:

$$N_{rcvd}[P_{exp}] \leftarrow (Seq\_No + 1) \bmod M;$$
$$P_{exp} \leftarrow Next\_Path;$$

the cell is dequeued, and step (O1) is repeated;

(O3) if $(Seq\_No = N_{rcvd}[P_{exp}] + 1) \bmod M$, then a cell loss is declared and the cell is left at the head of the FIFO queue. The data structures are updated:

$$N_{rcvd}[P_{exp}] \leftarrow (Seq\_No + 1) \bmod M.$$
$$P_{exp} \leftarrow Prev\_Next\_Path.$$

and the control is returned to step (O1);

(O4) if Seq_No differs from the stored expected value by more than 1, a severe cell loss is declared leading to the invocation of the Advanced Frame Recovery Mode.

Figure 7 provides an example of the protocol operation. The solid arrows point to the next switching path to be used, whereas the dash arrows indicate the retransmitted pointer values. A unit gap in the per-subconnection sequence numbers, as is the case with switching path $P3$, allows to detect a single cell loss event. Although the next switching path pointer of the lost cell (dotted arrow) is missing, the split VC cell chain can be restored using the pointer value retransmitted with the next cell of the subconnection (wide dash arrow).

We observe that the use of the NEXT_PATH pointer requires the dispatching to be *predictive*, i.e., the input controller has to make the selection for the path of the next cell routing prior to the departure of the previous cell of the same subconnection, even if the next cell itself has not yet arrived.

4.3 Properties of the Protocol

In summary:

- If no losses occur and the differential delays between subconnections is finite, the protocol re-assembles cells in the same order as they were dispatched. Unambiguous re-assembly is ensured.
- The protocol *automatically recovers* the cell re-assembly chain if not more than one cell from the same subconnection is lost in a row.
- The protocol *detects* an occurrence of cell loss within a subconnection, if the number L of consecutively lost cells satisfies $1 \leq L \bmod M \leq M - 1$; if L is a multiple of M, the protocol fails to detect the cell loss.
- The protocol recovers incorrectly if the number L of consecutively lost cells in a subconnection exceeds M and $L \bmod M = 1$.

Finally, we observe that, in general, depending upon the availability of the control bandwidth, the recovery from L_r consecutive losses ($L_r < M$) can be achieved by transmitting, in the local header of each cell, the L_r most recent P_{next} pointers of the same subconnection.

5 Advanced Frame Recovery Mode

The Advanced Frame Recovery Mode is a robust extension of the Cell Ordering and Re-assembly Protocol based on the regular insertion of sequentially numbered checkpoints into the cell stream of the split VC's. The checkpoints partition each VC cell stream into *frames*. A portion of a frame dispatched via the same switching path is referred to as a *subframe*. The switching path (and the corresponding subconnection) that carries the first cell of a frame, becomes the *leader* of that frame. Since the cells of each split VC are dispatched in FIFO order, the cells belonging to different subframes can not interleave within a subconnection. Therefore, if the re-assembly algorithm aligns each subconnection at a subframe corresponding to the *same* checkpoint and correctly selects the leader, then the cell sequence can be restored from that point on. Regularity of checkpoint insertion does not necessarily imply periodicity; instead the checkpoints may follow, for example, the boundaries of the transport layer protocol frames.

The checkpoints are implemented with a *painted header*, a special form of the local routing header which is assigned to the first cell of each non-empty subframe. The painted header contains the frame sequence number and the leader flag (alternatively, the leader flag can be eliminated by always sending the leader on one pre-negotiated path). The length of the frame sequence number field depends on the maximum number of frames that SCIMA must tolerate to loose, and is typically small. With careful design, it is possible to superimpose the entire painted header on existing fields in the local header, thus not adding additional overhead. When the Advanced Frame Recovery Mode is invoked on a split VC, the output SCIMA controller repeatedly *flushes* the subconnection re-assembly queues $Q_{FIFO}[1:K]$ until a painted header cell is encountered. Flushing is repeated until all K queues are aligned at the beginning of the same frame, or are empty, and exactly one leader is found among the painted headers. At this point P_{exp} is reset and the sequence numbers $N_{rcvd}[1:K]$ are re-initialized. Loss of a painted header cell is not catastrophic as the controller will merely try to align all the subconnections at the beginning of the next frame.

6 Adaptive Cell Dispatch Algorithm

Finally, we describe the algorithm which dispatches the cells to the individual paths and performs dynamic load balancing according to the congestion state of the paths. The algorithm runs in the input controllers and serves two purposes: first, for every split VC, it predictively computes the sequence of switching paths to which the cells have to be routed; second, for every port in the core switch, it performs scheduling between the subconnections and the unsplit portion of the traffic destined to that port.

For the split connections C_r, $r = 1, \ldots, S$, the algorithm distributes the traffic load in proportion to the assigned split weights $W^{(r)}$, irrespectively of the cell arrival process. It also accounts for the backpressure information provided by the input ports of the core switch, selectively for each subconnection. As mentioned above, the backpressure mechanism is not intended to decrease the losses for the split VC's, but rather to transfer the losses to the input controllers, before any dispatching decision has been made. The unsplit VC's, which are not subject to backpressure, may still experience congestion and loss in the input ports of the core switch.

The actual dispatching mechanism consists of a two-level Weighted-Round-Robin (WRR) scheduler, whose structure is schematically shown in Fig. 8 (for simplicity, as discussed in Section 4 above, we assume that a split connection is always associated with a single switching path per port):

- A *first level* WRR scheduler is instantiated for every split connection C_r using the split weight vector $W^{(r)}$, to dispatch the cells of the VC to the switch ports in the group in proportion to the split weights. For each split VC, this stage consists of a single *pre-split FIFO queue* to store the cells before they are committed to a particular port in the core switch, and one *subconnection buffer* to store cells immediately before they are sent to a selected port. Since it is desirable to commit a cell to a specific port as late as possible, each subconnection buffer typically only needs to store a single cell.
  In order to be consistent with our cell ordering and re-assembly scheme described above, the first level scheduler actually acts on the next pointer P_{next} rather than on the cell itself (in other words, commits the next cell rather than the current cell). In selecting where the next cell is going to be dispatched, the scheduler takes into account the backpressure information for that VC, and does not send cells to paths experiencing congestion, redirecting that traffic to other paths. Since the scheduler is actually scheduling "next cells", the effect is that of an increased latency in backpressure having effect on the traffic, and therefore the core switch needs to reserve one cell buffer space per subconnection to absorb such latency.
- A *second level* WRR scheduler is associated with each port i of the core switch attached to the input controller, and arbitrates among all the unsplit VC's and the split subconnections destined to port i. In this scheduler, the subconnections corresponding to a split connection C_r are served with weights $w_i^{(r)} \times B^{(r)}$, where $w_i^{(r)}$ is the split weight of the subconnection and $B^{(r)}$ is the requested bandwidth of connection C_r. In addition, the scheduler serves a single *unsplit FIFO port queue* where all the cells of the unsplit VC's destined to that port are directly queued on arrival. The weight of this queue is equal to the sum of the bandwidths of the unsplit VC's destined to port i. This second level of the scheduler is not affected by backpressure information.

In practice, this hierarchical WRR mechanism can be modified to synchronize the two levels of the scheduler.

7 Conclusions

In this paper, we have generalized the concept of inverse multiplexing to the context of switched ATM connections. The scheme allows to use a $kN \times kN$ ATM switch with ports operating at line rate R to implement an $N \times N$ switch with ports operating at line rate kR. This avoids large re-design costs of the switching equipment when interfacing high-capacity trunks to existing switch cores having lower-rate ports.

Whereas the conventional Inverse Multiplexing for ATM (IMA) performs inverse multiplexing of the entire ATM traffic

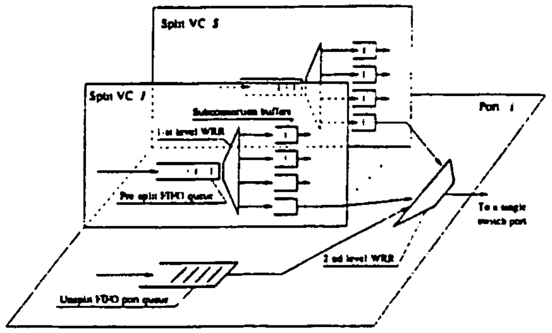

Figure 8: Input SCIMA controller: two-level cell dispatch mechanism.

stream as a whole, using synchronous techniques for differential delay compensation, and is therefore suited for bufferless and lossless point-to-point transmission links, the Switched Connection Inverse Multiplexing for ATM (SCIMA) presented in this paper is specifically designed to operate on a per-VC basis in presence of cell loss, congestion, and large unpredictable differential delays. We have designed a set of supporting protocols, which includes split connection cell ordering and re-assembly, adaptive cell dispatching with prediction, and advanced frame recovery in the presence of severe faults.

References

[1] Interoperability Requirements for $N \times 56/64$ kbit/s Calls, Version 1.0, *BONDING Consortium*. Sept. 1992.

[2] P. H. Fredette, "The Past, Present, and Future of Inverse Multiplexing," *IEEE Communications*, pp. 42-46, April 1994.

[3] J. Duncanson, "Inverse Multiplexing", *IEEE Communications*, pp. 34-41, April 1994.

[4] ATM Forum Inverse Multiplexing for ATM Specification, Version 1.0, July 1997.

[5] F. M. Chiussi, J. G. Kneuer, and V. P. Kumar, "Low-Cost Scalable Switching Solutions for Broadband Networking: The ATLANTA Architecture and Chipset," *IEEE Communications*, pp. 44-53, Dec. 1997.

[6] J. Turner, "Design of a Broadcast Packet Switching Network," *IEEE Trans. on Communications*, pp. 734-743, June 1988.

[7] H. S. Kim and A. Leon-Garcia, "A Self-Routing Multistage Switching Network for Broadband ISDN," *IEEE J. Sel. Areas in Commun.*, pp. 459-466, April 1990.

[8] I. Widjaja and A. Leon-Garcia, "The Helical Switch: A Multipath ATM Switch Which Preserves Cell Sequence," *IEEE Trans. on Communications*, Vol.42, no.8, pp. 2618-2629, Aug. 1994.

[9] E. Desmet, B. Steyaert, H. Bruneel, and G. H. M. Petit, "Performance Analysis of a Resequencing Unit in a Multipath Self-Routing Switch Fabric," *Proc. 14th Int. Teletraffic Congress*, pp. 611-621, Amsterdam, Netherlands, 1994.

[10] J. Frimmel, "Inverse Multiplexing: Tailor-made for ATM," *Telephony*, pp. 28-34, July 1996.